Self-Cleansing Systems for Intrusion Containment

Yih Huang
Computer Science Department
George Mason University
Fairfax, VA 22030
(703) 993 1540
huangyih@cs.gmu.edu

Arun Sood
Computer Science Department
George Mason University
Fairfax, VA 22030
(703) 993 1524
asood@cs.gmu.edu

ABSTRACT

In this paper, we discuss the application of high-availability computing systems to intrusion containment. Intrusion Management Systems (IMS) serve to protect complex computer systems from unauthorized intrusions. The traditional IMS approaches rely on intrusion prevention and detection, followed by implementation of intrusion resistance procedures. A key assumption of a traditional IMS is that it is possible to detect all intrusions. We believe that the sophistication and rapid evolution of information warfare require the more pessimistic assumption that undetected intrusions will occur and must be guarded against as well.

Our approach, called *Self-Cleansing Intrusion Tolerance* (**SCIT**), pushes the concept of high-availability computing one step further. In a SCIT system, a server is periodically assumed to have "failed," namely, comprised by undetected intrusion. Consequently, the server is brought off-line for cleansing and integrity checking while a backup takes over. Indeed, it is more appropriate to see a SCIT system as two mirror servers working alternatively than as a primary server and its backup. In this paper, we define the concept of SCIT, present our experiences in building a SCIT firewall prototype, and discuss the future work in more advanced SCIT servers.

Keywords

high-availability computing, computer security, intrusion containment, self-cleansing systems.

1. INTRODUCTION

Computer systems are becoming more complex and are increasingly vulnerable to cyber warfare. Typical (traditional) Intrusion Management Systems (IMS) are based on intrusion prevention and detection followed by implementation of intrusion resistance procedures [1,2]. The latter generally includes intrusion tracking and subsystem isolation. Such an IMS approach relies heavily on the ability to detect intrusion events in the first place.

We however make the pessimistic but realistic assumption that not all intrusion activities can be detected and blocked (at least not in a timely manner to avoid significant damage) and seek technologies to build secure systems which constantly assume the compromise of the system and perform self-cleansing, regardless of whether intrusion alarms actually occur or not. We argue that such an assumption is appropriate given the sophistication and rapid evolution of information warfare. It is especially important for critical distributed computing systems: To achieve the highest level of security, we must not be overconfident in either our knowledge of enemy tactics and technologies or our capability to fend off *all* attacks.

One implementation of self-cleansing involves rebooting the subsystem from a trusted storage device followed by, if necessary, system recovery, checkpoint, rollback, and data integrity checking routines. (A trusted storage device can be either a read-only storage device or any nonvolatile storage where information is cryptographically signed.) System availability is achieved by means of redundancy, that is, a second mirror system is brought online to provide services. In this way, SCIT can be considered as a branch of high-availability computing [3,4]: In a highly available system, sufficient hardware redundancy is built into the system so that a backup can immediately replace a failed system. In SCIT, the switching from one system to its mirror is not only triggered by failures; it is a regular routine designed to root out undetected intrusion activities.

To illustrate, we apply the SCIT approach to firewalls. Here we assume that the decision of whether to drop a packet is strictly made on a per-packet basis.[1] Firewalls are widely used to block undesirable, potentially hostile packets at the entry to a secure site. A successful and unnoticed firewall subversion will leave the door to the site open, exposing the internals of the victimized site to the outside

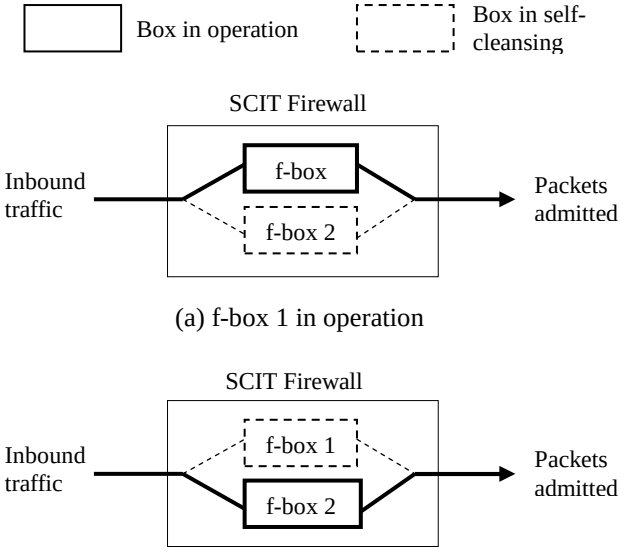
This research is supported by US Army's Telemedicine and Advanced Technology Research Center (TATRC), under contract number DAMD 17-01-1-0825.

[1] The cases of *stateful* firewalls, which maintain state information of ongoing TCP connections, will be more involved. Also, we do not consider proxy servers as part of the firewall.

world. As seen in Figure 1, we simply use two identical firewalls, called *f-boxes* in the figure. When one f-box is working, the other box will be performing self-cleansing by rebooting itself. Assuming that rebooting is from read-only devices, a rebooted f-box will be in a clean state and can perform packet filtering needed to protect the site. As such, even if the enemy managed to break into one f-box, its control over that box is limited to one self-cleansing cycle. More complicated SCIT systems will be discussed later.

Figure 1. SCIT Firewall

System self-cleansing limits the amount of time that a successful intruder has to stay in the system and inflict



(a) f-box 1 in operation

(b) f-box 2 in operation

damages. The longer this *Intruder Residence Time* the greater the damage and loss. We anticipate that the loss curve will be an S-curve of the form in Figure 2. If the Intruder Residence Time is less than the low loss threshold, then the cost of the intrusion is low, while an Intruder Residence Time greater that the high loss threshold will lead to near max loss. The steep slope between the two thresholds indicates that it is necessary to limit the Intruder Residence Time to less than low loss threshold. The low loss threshold reflects the reality that it takes a certain time window for a hacker to be able to issue malicious commands, exploit backdoors, install Trojan horse programs, and/or steal/destroy data. A conservative estimate of the low loss threshold is in the range of minutes. Although there is no hard data for building the loss curve, there are reports that can help the process of building such a curve. For example, in [5] it is reported that in the context of on-line banking, security experts believe that a theft of $5,000 to $10,000 can be carried out over a few weeks, while larger losses up to $1 million are likely to take four to six months. In this context it is emphasized that the loss curve must account for the possibility that the *Intruder Residence Time* is spread over more than one successful breach of the system.

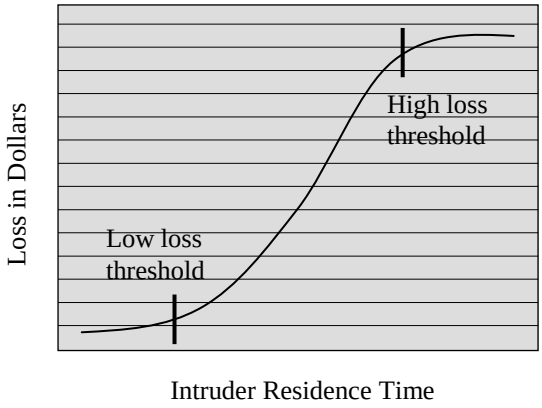


Figure 2: Loss curve: Loss in dollars vs. Intruder Residence Time in minutes

Lastly, we point out that SCIT complements and strengthens existing intrusion prevention and detection technologies [1,2]; we do not eliminate the use of the current intrusion management systems, but rather add another layer of defense, extending the idea of system "defense-in-depth" through periodic system cleansing. The effectiveness of SCIT depends on fast self-cleansing cycles, restricting the attackers to a very short time window to breach the system and cause harms.

The remainder of the paper is organized as follows. In Section 2, we present our experiences in building a prototype of SCIT firewalls. In Section 3, we discuss the feasibility of "SCIT-izing" more complicated systems, such as file servers. We give concluding remarks and outline future work in Section 4.

2. SCIT FIREWALLS

We chose firewalls as the first application of SCIT. The rational is twofold. First, the operation of stateless firewalls lends itself to SCIT, owing to the relative ease for a backup or mirror system to take over without disrupting ongoing traffic. Second, firewalls form the first line of defense for many private networks and thus are obvious targets of intrusion attacks. Strengthening the defense of firewalls significantly reduces the risk of security breaches in the whole network. The applications of the SCIT approach to more complex systems, such as NFS, DNS, and Web servers, will be discussed later.

Our testbed is based on the Virtual Machine software from VMWare, Inc. [6]. The virtual machine technology enables multiple *guest operating systems* to be installed and executed on top of a *host operating system*. In our demo, a client uses the Explorer on Windows 2000 to surf the web.

The client Windows is protected by SCIT firewalls. Both the client Windows machine and its SCIT firewalls are implemented as *virtual machines*. The underlying host machine is a Pentium 4 PC running RedHat 7.2 Linux. VMWare supports not only virtual machines but also *virtual networks*, which are emulated switched Ethernet networks. As seen in Figure 3, we use this feature to build two virtual networks, one with subnet ID 192.168.181.0/24 and the other 192.168.202.0/24. The first subnet connects the host system to the two firewalls. Both firewalls use a specialized version of RedHat 7.2 Linux. The latter subnet connects the firewalls to the client Windows. Inbound traffic are received by the host and relayed to one of the firewalls, which filters and relays the traffic to the client Windows. A third subnet, 192.168.200.0/24, is used by the two firewalls to probe each other. To enable the client Windows to communicate with the public Internet, its private IP address is translated to a public IP address by IP Masquerading, a form of network address translation that is supported by Linux kernels.

When a newly cleansed firewall is ready for operation, it must take over the IP addresses used by the presently running firewall. The firewall achieves this by issuing Gratuitous ARP messages using the Fake package [7]. The firewall rules are implemented in IPCHAINS. A shell script that executes the following steps controls the operations of each firewall. In the script, the first step is executed immediately after the underlying firewall has completed rebooting.

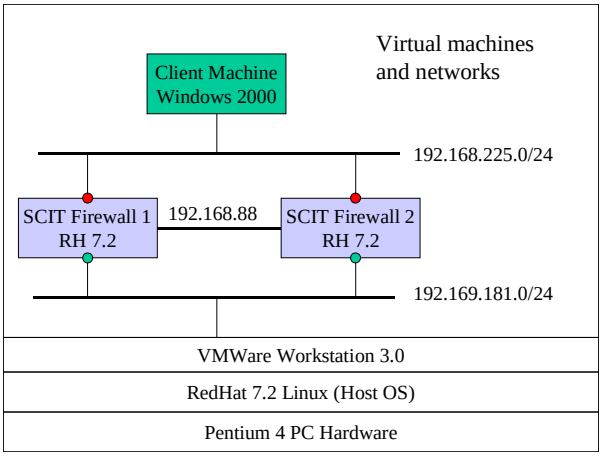


Figure 3. SCIT firewall prototype

1. Setup firewall rules using the `ipchains` command. Start traffic filtering and relaying in the background. The two tasks will continue to be performed in the background until the machine is shutdown for rebooting in Step 5.
2. Broadcast on subnets 192.168.225.0/24 and 192.168.181.0/24 Gratuitous ARP messages to announce the ownership of firewall IP addresses. This step lasts 10 seconds, one ARP message per second.
3. Wait for 10 seconds. This delay gives the other firewall extra times to detect the activities of this firewall and to reboot.
4. This firewall now assumes the other firewall is rebooting. It sends ping messages periodically over the 192.168.88.0 network to probe the other firewall, until a reply message is received. The receipt of a reply indicates the other firewall has completed self-cleansing and is ready to take over.
5. Reboot (and thus return to step 1 after completion).

From the client inside the SCIT firewall, the user opens a browser window and begins web surfing. We open two additional windows indicating which of the firewalls is operational. In the screen capture shown in Figure 4, these correspond to the black background windows on the top of the screen. The left-upper window shows that the presently running firewall is probing (unsuccessfully) the other firewall. That is, the firewall is executing the Step 4 given above. The right-upper window shows the booting message of the second firewall. The client Windows 2000 displays the CNN home page at the bottom. The page is, of course, obtained through the SCIT firewalls.

In general, for a typical HTML encoded web page, a firewall switch is barely perceptible. Examination of the trace sometimes shows occasional losses of packets when switching firewalls, but it appears that the retransmission of the packets is fast enough that the user cannot perceive the difference. Indeed, the setup is good enough for our own production uses – our research assistant regularly uses the setup for emails and web serving. With the setup, we observed self-cleansing cycles in the vicinity of 90 seconds.

Finally, it is worth pointing out that the self-cleansing of a firewall in the above steps comprises merely rebooting it. Assuming that the firewall is booted entirely from read-only storage, rebooting is sufficient to bring it to a clean state. While this assumption is reasonable for relatively simple devices like firewalls, in general cases more involved self-cleansing procedures are needed. For instance, using a tool called Tripwire, a system audit can be carried out after rebooting to check the integrity of system files [8].

3. A DISCUSSION OF SCIT SERVERS

We have shown a prototype design of SCIT firewalls. In this section we discuss the possibilities and difficulties of extending the concept to various type of servers in distributed computing environments. We call this task the "*SCIT-ization*" of servers.

1. Stateless servers are relatively straightforward to SCIT-ize. By stateless we mean the server does not

have to keep track of in memory the outcomes of previous tasks in order to carry out new tasks. NFS is a prominent example of stateless systems. Notice that dependences on the previous outcomes maintained in nonvolatile storage can be managed by SCIT, for mirror systems can share the storage. Storage sharing can be achieved by, for example, a SCSI bus in a small-scale system or a system-wide network (SAN) in a large cluster.

2. Servers that handle short sessions are relatively straightforward to SCIT-ize. Such servers are typically transaction oriented and process request-and-response types of tasks. Examples include DNS servers, some database servers, and certification servers. Telnet, FTP, and many application proxy servers are examples of long-session servers. A long session in a SCIT system needs to be migrated to a mirror system in the middle of the session. The task involves at a minimum moving endpoints of TCP connections on the fly and is unfeasible with the standard TCP. We will not further consider long-session servers for the time being.
3. Servers that manage static or semi-static data are relatively easy to SCIT-ize. A DNS, LDPA, or certification server, for instance, handles datasets that are typically small and infrequently changed. Static, small datasets enable efficient data mirroring and thus facilitates the construction of identical servers to operate alternatively. Due to the critical roles played by DNS and certification servers, SCIT technologies specifically developed for these servers further strengthen overall system security.

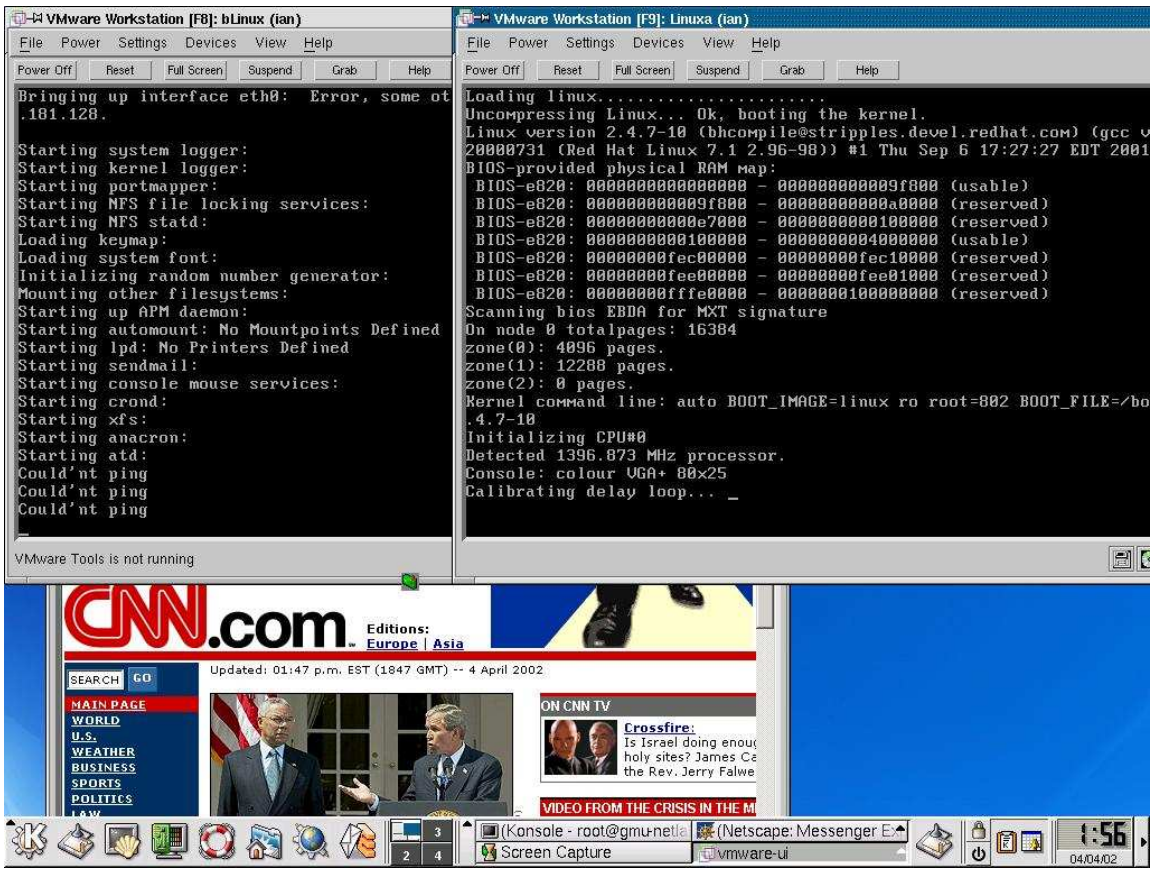


Figure 4. Screen capture of the SCIT firewall prototype

Indeed, we do not expect that the concept of SCIT be compatible with all types of computing systems. We do believe that SCIT is applicable to many important servers in distributed computing environments or Internet services, such as file systems, web servers, DNS servers, and certification services. The most important challenge in our future research is to design efficient SCIT architectures for these servers. To conclude this section, we present the

blueprint of our next SCIT system, a SCIT NFS server. The design is based on a high-availability file server architecture discussed in [9,10]. As shown in Figure 5, two server machines connect to a SCSI bus to share a SCSI hard drive that stores the file system data. Similar to the SCIT firewalls, the two server boxes must share one IP address using the technique described earlier. Also, a second set of network interfaces is used by currently running server to probe the status of the other server. At the time of this writing, we are implementing the design using the virtual machine technology.

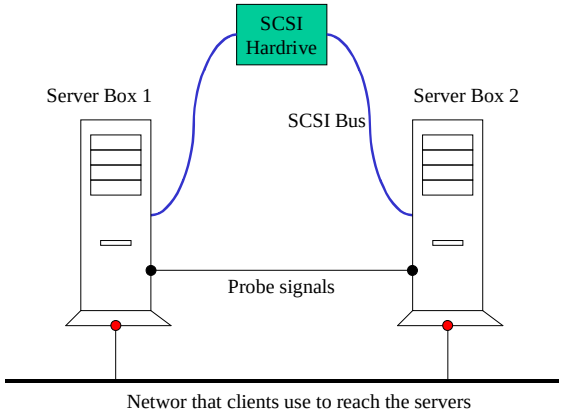


Figure 5. The blueprint of a SCIT NFS Server.

4. CONCLUSION

We have presented a novel application of high-availability computing, namely, intrusion containment. Our SCIT approach uses multiple, identical servers to execute in turn, allowing off-line servers to be checked for integrity and cleansed to return to a clean state. These self-cleansing activities occur periodically, regardless the presence/absence of intrusion alarms. As such, SCIT provides a defense against unknown or severe attacks that defeat the intrusion detection system. The effectiveness of SCIT depends on fast self-cleansing cycles, restricting the attackers to a very short time window to breach the system and inflict damages. The cost of hardware redundancy in SCIT systems can be avoided by using the virtual machine technology, as demonstrated in our SCIT firewall prototype.

At this early stage, we investigated only one self-cleansing method in detail, that is, rebooting followed possibly by data integrity checks and system audits. However, we envision many layers of cleansing activities in an ultimate SCIT system. In addition to rebooting the servers, one can kill and re-launch the server daemon. This process-level cleansing imposes less overhead, compared to system rebooting. Yet another system cleansing method is to re-load dynamic kernel modules, in the attempt to clean up those kernel codes potentially contaminated by hostile communications. With self-cleansing activities occurring at several levels of the system and at different frequencies, SCIT makes it very difficult for attackers to cause actual harms, even if they are able to penetrate existing intrusion defenses.

References

[1] M. Bishop, "Vulnerabilities Analysis," Recent Advances in Intrusion Detection, September 1999.

[2] Common Intrusion Detection Framework, http://www.gidos.org/.

[3] High-Availability Linux Project. Home page http://linux-ha.org/.

[4] Peter S. Weygant, *Clusters for High Availability*, Prentice Hall, 1996.

[5] Sandeep Junnarkar, "Anatomy of a hacking", available at http://news.com.com/2009-1017-893228.html, May 2002.

[6] VMware Inc. Home page http://www.vmware.com/.

[7] The Fake package. Home page http://vergenet.net/linux/fake/.

[8] Gene H. Kim and Eugene H. Spafford, "Writing, Supporting, and Evaluating Tripwire: A Publicly Available Security Tool," in *Proceedings of USENIX Applications Development Symposium*, (Toronto, Canada), April 1994. Also see http://www.tripwire.com/.

[9] Steve Blackmon and John Nguyen, "High-Availability File Server with Heartbeat," *System Admin, the Journal for UNIX Systems Administrators*, vol. 10, no. 9, September 2001.

[10] Richard Rabbat, Tom McNeal and Tim Burke, "A High-Availability Clustering Architecture with Data Integrity Guarantees," Proceedings of IEEE International conference on Cluster Computing, pages 178 – 182, (Newport Beach, California) October, 2001.